CS 6410: Compilers

Fall 2019

Project Introduction
Acknowledgment: Project assignment prepared by Hal Perkins, faculty member of the University of Washington, Allen School of Computer Science and Engineering

Posted on Tuesday, September 24, 2019
Instructor: Tamara Bonaci
College of Computer and Information Science
Northeastern University – Seattle

1 Project Overview

The course project is to implement the MiniJava language specified in the Appendix of *Appel's Modern Compiler Implementation in Java, 2nd edition*, and described on the MiniJava web site. MiniJava is a subset of Java, and the meaning of a MiniJava program is given by its meaning as a Java program.

1.1 About MiniJava

– Compared to full Java, a MiniJava program consists of a single class containing a static `main` method followed by zero or more other classes. There are no other static methods or variables.
– All classes in a MiniJava program are included in a single source file. The only types available are `int, boolean, int[]` and reference types (classes).
– Classes may extend other classes, and method overriding is included, but method overloading is not.
– You may assume that there is no predefined `Object` class, and that classes defined without an extends clause do not implicitly extend `Object`.
– All methods are value-returning, and must end with a single return statement.
– `System.out.println(...);` is a statement, and can only print integers - it is not a normal method call, and does not refer to a static method of a class.
– The only other available statements are `if, while`, and `assignment`.
– In addition to these, there are other simplifications to keep the project size reasonable.

The full MiniJava grammar is given on the project web site, and in the Appendix of *Appel's Modern Compiler Implementation in Java (2nd ed)*. Please take a look at the grammar carefully to see what is, and is not included in the language subset. You should implement full MiniJava as described there, **except that you do not need to implement nested /* comments */**. (You need to implement /* */ comments, but you do not need to allow them to nest, i.e., the first */ terminates any open comment regardless of how many /* symbols have appeared before it, as in standard Java. You also need to implement // comments.)

There are two symbols in the grammar that are not otherwise specified. An `<IDENTIFIER>` is a sequence of letters, digits, and underscores, starting with a letter. Uppercase letters are distinguished from lowercase. An `<INTEGER_LITERAL>` is a sequence of decimal digits that denotes an integer value.

2 Implementation

The default implementation platform for your project is **Java** 8 with the **JFlex/CUP** scanner/parser tools. You are free to use any development platform, and environment that you wish, but your resulting project should build and run using the provided Apache Ant build file.

If you wish to use another implementation language for the project (for example, C#, Haskell, C++, or another suitable language with appropriate scanner/parser tools and libraries), please discuss that with me, so that we make sure that your choice is a feasible.

3 Project Components

The project will be worth 100 points, and it will consist of several components. Please note that *most components will, in some way, depend on the functionalities of earlier built components.* The components, and their corresponding points breakdown, are as follows:

Part 1: Scanner (20 points)
Part 2: Parser and Abstract Syntax Tree (20 points)
Part 3: Semantics (20 points)
Part 4: Code generation (20 points)
Part 5: Additions and extensions to a compiler (10 points)
Part 6: Project report (10 points)

4 Resources

Below are the links to some online resources that you might find useful for your project:

1. MiniJava project web page, and MiniJava grammar (resources from Appel's compiler book)
2. JFlex lexical analyzer generator, and CUP LALR parser generator (including manuals for JFlex and CUP). I suggest that you use v11b of CUP since that allows Java generic types in semantic actions. JFlex and this version of CUP are included in the starter code.
3. The Java Language Specification from Sun, including a LALR(1) grammar for full Java. Check here if you are not sure how something should work.
4. **x86-64 resources**:
   - x86-64 Instructions and ABI (handout for University of Chicago CMSC 22620 by John Reppy)
   - x86-64 Machine Level Programming by Bryant and O'Hallaron (CMU; earlier version of Sec. 3.13 of Computer Systems: A Programmer's Perspective, 2nd ed.)
   - Web resources to Bryant'O'Hallaron's textbook Computer Systems: A Programmer's Perspective(links to various x86/x86-64 resources)
   - gdb summary (direct link) (One they didn't include: display/i $ rip will show you the next instruction every time you step or reach a breakpoint).
   - SSE floating point
5. gdb documentation
6. UW gdb Reference card (direct link)
7. Gnu assembler documentation